Naval Research Laboratory
Stennis Space Center, MS 39529-5004

Parallel and Vector Adaptations of the Underwater Finite Element Parabolic Equation Model

ROBERT A. ZINGARELLI

Acoustic Simulations and Tactics Branch
Center for Environmental Acoustics

August 18, 1993

DTIC
S ELECTE
SEP 2 1 1993
E D

93 9 20 074

93-21880

REPORT DOCUMENTATION PAGE

Form Approved
OBM No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 18, 1993	Final

4. TITLE AND SUBTITLE

Parallel and Vector Adaptations of the Underwater
Finite Element Parabolic Equation Model

6. AUTHOR(S)

Robert A. Zingarelli

5. FUNDING NUMBERS

Job Order No.	571506003
Program Element No.	0602435N
Project No.	RJ35O01
Task No.	JOV
Accession No.	DN250019

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Center for Environmental Acoustics
Stennis Space Center, MS 39529-5004

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/FR/7181--93-9430

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report describes optimization techniques for several vector and parallel versions of FEPE, a finite element underwater parabolic equation model code. Algorithms are presented for solving linked tridiagonal systems of equations, when the chief computational bottleneck is in FEPE on parallel, massively parallel, and vector machines. Sample execution times for a standard benchmark problem are also given. A broadband version of FEPE, which is highly efficient on vector computers and can be combined with parallel algorithms for even higher performance is presented.

14. SUBJECT TERMS

underwater acoustics, active acoustic propagation, surface scattering, ambient noise

15. NUMBER OF PAGES
9

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Same as report

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

CONTENTS

PARALLEL AND VECTOR ADAPTATIONS OF THE UNDERWATER FINITE ELEMENT PARABOLIC EQUATION MODEL

1.0 INTRODUCTION

A finite element parabolic equation (FEPE) model code was developed and optimized for a conventional single-instruction/single-data (SISD) computer [1]. Subsequent acquisition of several supercomputers by the Naval Research Laboratory (NRL) and the Naval Oceanographic Office (NAVOCEANO), coupled with increased demands for performance by this code, spurred investigation into optimization of FEPE on supercomputers. This report briefly describes FEPE and why, in its SISD optimized form, it does not execute efficiently on supercomputers; the algorithms used in optimizing FEPE for several supercomputer architectures; and a brief users guide to optimized and verified versions of the code.

2.0 FEPE AND SUPERCOMPUTERS: A PEDESTRIAN'S GUIDE

FEPE is fully documented for users in the "FEPE User's Guide" [1], and the algorithms and methods used are described in more detail in Ref. 2. Briefly, at each range step FEPE uses the Galerkin FE method to generate two linked tridiagonal sets of coupled equations (one equation for each vertical grid element, and a set of equations above and below the water-sediment interface), which it then solves by Gaussian elimination with backsubstitution. Typically, several hundred to a few tens of thousands of vertical gridpoints are needed for accurate solution of the PE, depending on source frequency and total depth of the problem. Parameters that incorporate fluid densities and sound speed profiles for water and sediment are precalculated and stored in matrices at the beginning of the program, and are updated as profiles change with range. These parameters are used to weigh each Galerkin step in the solution of the resulting tridiagonal system.

The raw number of floating-point operations in a typical run is split roughly equally between the Galerkin method weighting and the equation system solution steps; the matrix generation and update routines occupy no more than a few percent of the total execution time. The weighting loop is recurrence free and iterates once for each vertical gridpoint. The tridiagonal solver routine contains four loops (a forward and backward substitution for above and below the interface) of the form $x_i = a_i x_{i-1} + b_i$, which is a first-order linear recurrence and not directly susceptible to vector or parallel processing. A preliminary port of FEPE to a vector supercomputer [3] showed that more than 85% of the total execution time was spent in solving the tridiagonal equations in the SISD mode at roughly one-tenth of the machine's peak speed.

3.0 ALGORITHMS

Although FEPE was originally written and optimized for SISD machines where recurrence relations carry no performance penalty, concurrent modes of computer operation, single-instruction/multiple-data (SIMD), and multiple-instruction/multiple-data (MIMD) are able to indirectly solve

such recurrences by precalculating selected points in the iteration chain and subsequently working from these or by simultaneously accumulating the final values for all equations in an iterative manner.

3.1 Chunk Algorithms

By rewriting the original recurrence in matrix form:

$$\begin{bmatrix} x_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_i \\ 1 \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_i \\ 1 \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{i-1} & b_{i-1} \\ 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} a_1 & b_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ 1 \end{bmatrix},$$

$$(1)$$

we see that intermediate products of several matrices may be computed independently, and that these products may be multiplied sequentially with the starting value x_1 to give a set of known x_i's scattered along the recurrence which serve as starting points for multiple independent calculations. By allocating the work of calculating matrix products, starting points, and recurrence segments among N processors for an n-length recurrence, with the $n \bmod N$ extra points given to the first few processors, we arrive at the MIMD algorithm given in Ref. 4 and shown in Fig. 1. By allocating the first $n - n \bmod N$ elements to N processors and then leaving the few extras for subsequent SISD
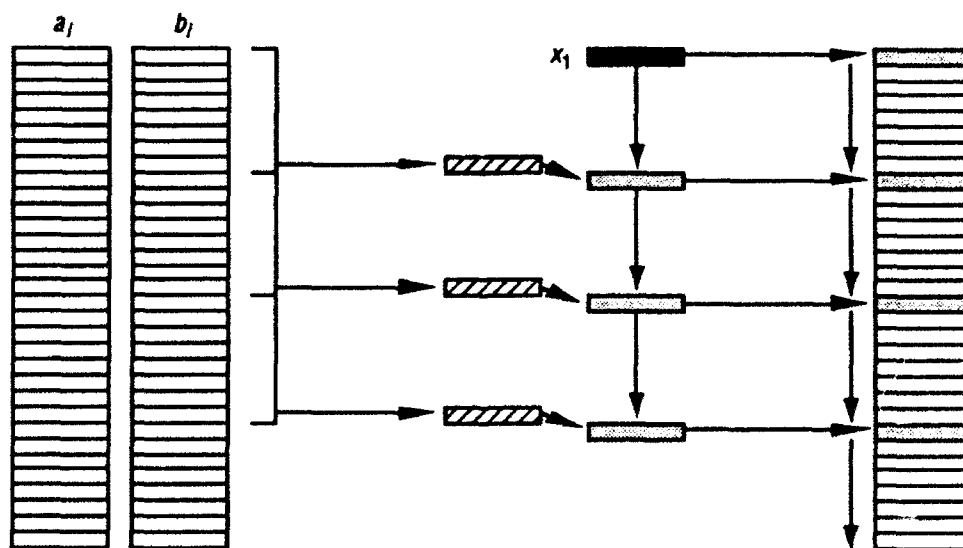

Fig. 1 — Diagram of MIMD chunk algorithm (m-chunk) showing flow of information. In first stage of computation (leftmost), partial products of coefficients are calculated in parallel. In second step (center), these partial products are used to serially precalculate intermediate starting points for final, parallel, iteration of recurrence relation (right).
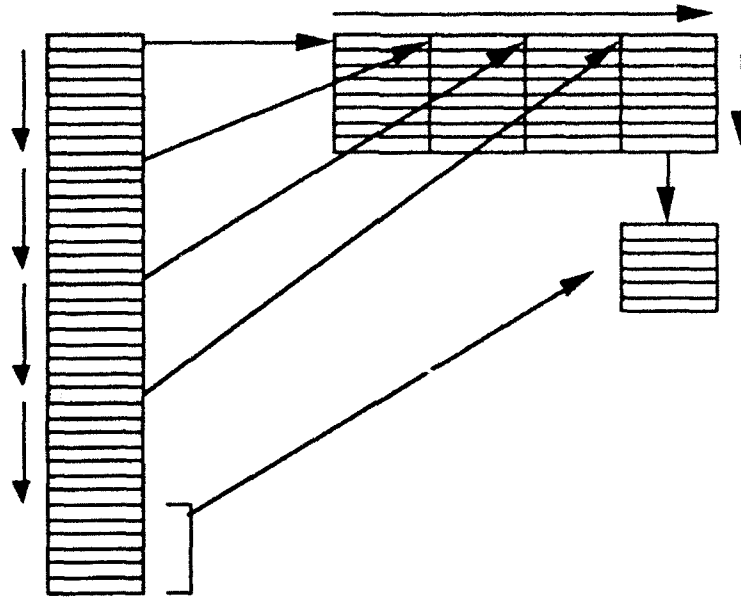
Fig. 2 — Rearrangement of final step in Fig. 1 to allow SIMD processing.

solution, we arrive at a related SIMD algorithm. This idea, applied to only the final step for clarity, is shown in Fig. 2. Because these two algorithms break up the problem into pieces, or chunks, and parcel the chunks out to processors, we will refer to them as m-chunk and s-chunk, respectively, for the MIMD and SIMD versions.

Execution times for the two algorithms, T_{MIMD} and T_{SIMD}, in units of time (T) to perform a floating-point operation on a single processor, are:

$$T_{MIMD} = \frac{5n}{P(N)} - 5 + 2N$$

$$T_{SIMD} = \frac{5n}{V(N)} - 5 + 3N,$$

(2)

where $P(N)$ is the MIMD-mode speed-up factor for N processors; usually this is linear in N but may be reduced by interprocessor communication times. $V(N)$ is SIMD-mode speed-up, which will be the same as $P(N)$ for a parallel-SIMD machine (pSIMD) or will reach some maximum value (typically about 10) for a vector-SIMD machine (vSIMD). The SISD execution time for the same type of processor is $2n$. Given an analytic form for $P(N)$ or $V(N)$ these times may be minimized to find the optimum number of processors and the maximum speed-up possible for a given n. For example, if $P(N) = N$, $N_{opt} = \sqrt{5n/2}$ and the maximum speed-up is $\sqrt{n/10}$. Figure 3 shows how performance varies with N for the MIMD algorithm for $n = 2000$.

3.2 Cyclic Reduction

An alternative method for computing this recurrence is the cyclic reduction algorithm discussed in Ref. 5. The recurrence can be rewritten as follows:
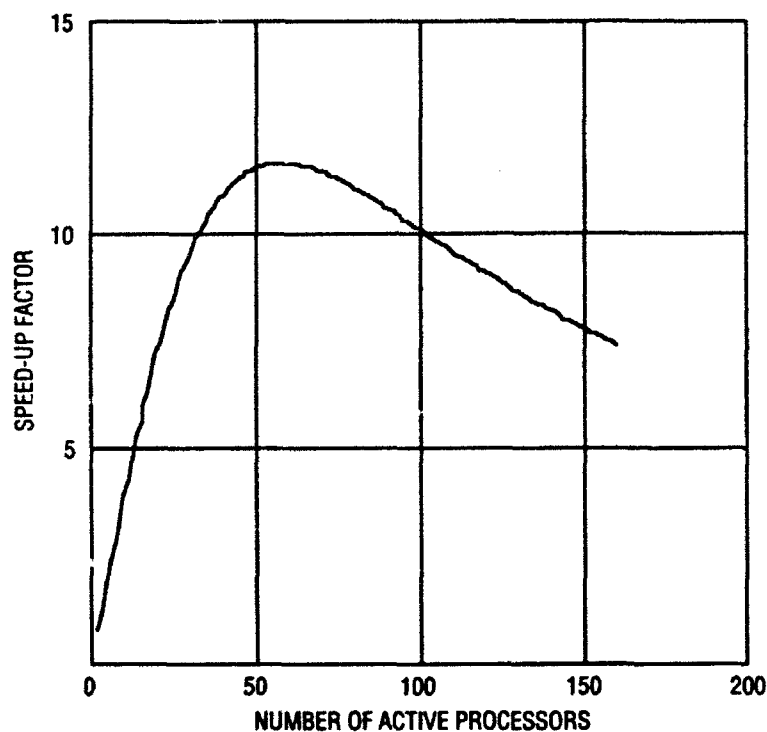
Fig. 3 — T_{SISD}/T_{MIMD} for 2000-element recurrence relation discussed in text, and 2 to 160 active processors. Communication and synchronization overhead has been neglected from this calculation (i.e., $P(N) = N$).

$$x_1 = b_1$$

$$x_2 = b_2 + a_2 x_1$$

$$x_3 = b_3 + a_3 b_2 + a_3 a_2 b_1$$

$$x_i = \sum_{j=1}^{i} b_j \prod_{k=j+1}^{i} a_k \tag{3}$$

With some manipulation, this can be reduced to:

$$x_{2i}(j) = x_i(j) + x_i(j-i) \cdot \prod_{k=j-i+1}^{j} a_k \tag{4}$$

where $x_i(j) = x_i(b_j, ..., b_{j-i+1}, a_j, ... a_{j-i+1})$.

Now define

$$M_i(j) = \prod_{k=j-i+1}^{j} a_k \quad \text{for } j \geq i$$

$$= \prod_{k=1}^{j} a_k \quad \text{for } j < i, \tag{5}$$

and Eqs. (4) and (5) can be reduced to:

$$x_{2i}(j) = x_i(j) + x_i(j-i) \cdot M_i(j-i)$$

$$M_{2i}(j) = M_i(j) \cdot M_i(j-i) . \tag{6}$$

By initially setting $x_i = b_i$ and $M_i = a_i$ for $1 \le i \le n$ and iterating Eq. (6) $\log_2 n$ times (that is, by starting with $i = 1$ and doubling through to $i = n$) for $i + 1 \le j \le n$, a solution is found in $3n \log_2 n / P(N)$ on N processors. N may be as large as n, and if $P(N) = N$, then the total solution time is reduced to $3 \log_2 n$. If n is not an even multiple of 2, this algorithm still works but must be iterated to the next higher power of 2.

3.3 Broadband Techniques

If the same propagation path is to be used for many frequencies, the recurrence may be side-stepped entirely by computing the recurrences for all frequencies in parallel, or by looping over frequencies within each step on a vSIMD machine. This method has the advantage of not imposing any additional overhead within the code to compute intermediate results, but has the disadvantages that a user may not have enough frequencies to fill all processors and, since all frequencies are being run effectively in parallel, may require excessive memory. If not having enough frequencies to fill available processors is a problem, this technique can be combined with one of the chunk or cyclic reduction algorithms.

3.4 Split Solver Sections Executed in Parallel

A final optimization is possible by running the water column and sediment solvers in parallel. This will work only on MIMD architectures and will be efficient only if the sizes of the two pieces of the computation are comparable. In the cases of the serial and chunk algorithms, the computation sizes are linear in recurrence length, but with the cyclic reduction algorithm the computation sizes go as $\log_2 n$, which frequently will provide a close-enough match to make this technique worthwhile.

4.0 APPLICATIONS AND RESULTS

With four algorithms in hand, the solver routine for FEPE was rewritten in a variety of ways. The available machines were a Cray Y-MP/8128 and a Thinking Machines Corp. CM-200. The Y-MP has eight shared-memory processors, each of which may be operated in SISD or vSIMD mode. In turn, the processors may be operated independently or in MIMD mode. The CM-200 can be configured to operate on 8192 or 16,384 processors in a nearly pure pSIMD mode. We use the term "nearly pure pSIMD" because the CM-200 instruction set has masking operations that allow two different operations to proceed concurrently on different groups of processors, giving this machine a very limited MIMD capability.

The problem chosen was the penetrable wedge benchmark [6], with either 1 or 64 frequencies, with a 0.5-m depth step down to 2 km, which requires 4002 vertical gridpoints. The water column varies from 200 to 0 m in depth, so executing the two parts of the solver in parallel will make little difference in the chunk algorithms but could lower the execution time of the cyclic reduction algorithm by 43% due to the $\log_2 n$, rather than linear, time dependence of this algorithm.

Table 1 gives execution times per frequency for the presented algorithms applied to one Y-MP processor, eight Y-MP processors in MIMD mode, and 4002 CM-200 processors. Figure 4 shows the reciprocal of selected times from this table, along with reciprocal of charged processor times, to give an indication of the overall cost and speed in propagation paths computed per second.

Of the single-frequency calculations, the s-chunk algorithm on a single Y-MP processor proved to be the most cost effective. It ran the benchmark case 40% faster than the original version. The m-chunk algorithm, while fast and interesting, was not especially efficient because all calculations were carried out in scalar mode on eight vector processors. This algorithm would be better suited for parallel machines made up of scalar processors; various hypercube configurations and networked workstations come to mind here.

Table 1 — Wallclock Execution Times Per Frequency for Various Algorithms on Cray Y-MP/8 and CM-200 using 4002 Processors. Unsuitable Algorithms Marked "Not Applicable." Multifrequency Versions of s-chunk and the Cyclic Reduction Algorithm were Not Implemented (see text).

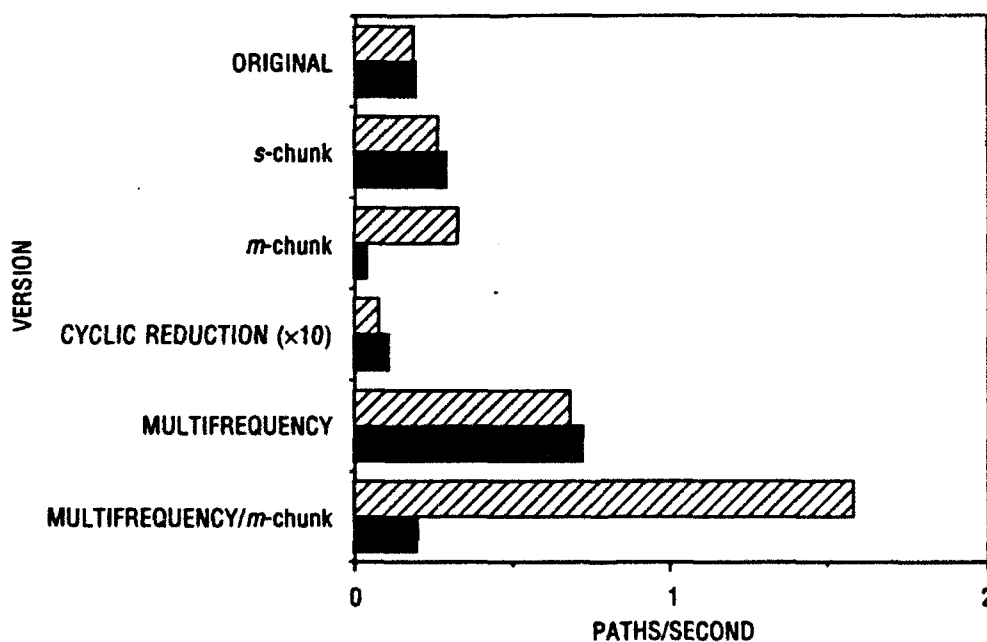| Machine	Original	s-chunk	m-chunk	Cyclic Reduction	Multifrequency	Multifrequency/ m-chunk
Y-MP	5.29	3.77	N/A	38.62	1.46	N/A
Y-MP/8	N/A	N/A	3.05	N/A	1.21	0.63
CM-200	N/A	N/A	N/A	108.44	N/A	N/A

Fig. 4 — Speed, in propagation paths per second for penetrable wedge benchmark, for general versions of FEPE. Shaded bars are paths per wallclock second; black bars are paths per processor second.

Overall on the Y-MP, the multifrequency versions quadrupled the performance, both in processor and total execution time, of the single-frequency versions. This improvement is due to the vectorized inner loop over frequency, allowing efficient use of the processors. The original serial recurrence and m-chunk solvers were rewritten using this frequency vectoring technique. The floating-point rates per processor for these two versions were in the 150–200 million per second range (Mflops), near the peak speed for this type of machine.

Because the cyclic reduction algorithm performed poorly on the Y-MP, it was implemented on only a single processor. Peak speeds were in the 50 Mflops range due to irregular-stride memory access. On the CM-200, this same algorithm performed at only 0.36% of its theoretical peak rate for 4002 processors, despite extensive hand-optimization. After lengthy investigation and discussion with NRL Connection Machine Facility personnel, we have decided that router communication between processors is the limiting factor in this type of calculation. Clearly, while interesting, the cyclic reduction algorithm is unusable for this problem without extensive machine-level coding or specialized hardware.

A combination of the s-chunk and multifrequency algorithms could run efficiently on the CM-200. For example, a problem with 4002 vertical gridpoints and 200 frequencies would optimally run on 16,334 processors with a potential speed-up factor over 3000. Unfortunately, the s-chunk algorithm would require extensive router communications between processors and would probably not reach such high speeds.

Of the various algorithms and machine combinations tried, the four most cost-effective versions ran on the Y-MP:

• FEPE64v: single-frequency s-chunk algorithm on one processor; 40% faster than original with scalar solver.

• FEPE64mfv: multifrequency vectored adaptation of original solver on single processor; achieves near-peak speed on Y-MP.

• FEPE64mfvp: same as mfv version but weighting loop parallelized as well; can achieve 40% greater speed than vector-only version but is about 10% more costly to run.

• FEPE64mfvc: multifrequency vectored m-chunk algorithm allows all eight Y-MP processors to work in parallel and vector modes.

Additionally, the solver routine for the single-frequency m-chunk version, solvep, has been left in place as a guide for understanding the solver in FEPE64v and as a frame for adaptation to other MIMD machines.

The input file remains unchanged for the single-frequency version, however, the second line of the input file for the multifrequency version has been modified

from		frequency	source depth	receiver depth	
to	number of frequencies	first frequency	last frequency	source depth	receiver depth

Beyond this change, the input file is as documented in Ref. 1.

5.0 SUMMARY

FEPE has been optimized for SIMD and MIMD machines with varying degrees of success. Examination of the original code revealed that most of the execution time is spent solving the recurrence $x_i = a_i x_{i-1} + b_i$. A chunk algorithm that makes this recurrence parallelizable was adapted for SIMD machines. The cyclic reduction algorithm performed poorly for the test cases used. Vectoring over frequencies was shown to be an efficient way of circumventing the recurrence bottleneck when broadband solutions are desired, and combined well with the m-chunk algorithm on an MIMD machine made up of shared-memory SIMD processors.

6.0 ACKNOWLEDGMENTS

7.0 REFERENCES

[1] M. D. Collins, "FEPE User's Guide," Naval Research Laboratory, Stennis Space Center MS, NORDA Tech. Note 365, 1988.

[2] M. D. Collins, "Benchmark Calculations for Higher-Order Parabolic Equations," *J. Acoust. Soc. Am.* **87**, 1535–1538 (1988).

[3] R. A. Zingarelli, "PE Codes: Supercomputer vs. Workstation Benchmarks," Naval Research Laboratory, Stennis Space Center MS, NRL/MR/7181--93-7015, January 1993.

[4] S. Brawer, *Introduction to Parallel Programming* (Academic Press, San Diego, CA, 1989), pp. 200–211.

[5] H. S. Stone, "An Efficient Parallel Algorithm for the Solution of a Tridiagonal Linear System of Equations," NASA Report 5424, 1971.

[6] F. Jensen and C. M. Ferla, "Numerical Solutions of Range Dependent Benchmark Problems in Ocean Acoustics," *J. Acoust. Soc. Am.* **87**, 1499–1510 (1988).